Theoretical Computer Science 100 (1992) 205–221
Elsevier

Polynomial-time 1-Turing reductions from # PH to # P

Seinosuke Toda

Department of Computer Science and Information Mathematics, University of Electro-communications, Chofugaoka, Chofu-shi, Tokyo 182, Japan

Osamu Watanabe

Department of Computer Science, Tokyo Institute of Technology, Ookayama, Meguro-ku, Tokyo 152, Japan

Communicated by J. Díaz
Received July 1989
Revised May 1990

Abstract

Toda, S. and O. Watanabe, Polynomial-time 1-Turing reductions from # PH to # P, Theoretical Computer Science 100 (1992) 205–221.

In this paper, we investigate relative complexity between # P and other classes of functions. Our particular interest is to compare # P with # PH and with PFH by using polynomial-time reducibility and to demonstrate that a weaker notion of polynomial-time reducibility is sufficiently powerful for reducing # PH functions to # P functions. Our main result is stated as follows: Every function in # PH is polynomial-time 1-Turing reducible to some function in # P. That is, $\#\mathrm{PH} \subseteq \mathrm{PF}^{\#\mathrm{P}[1]}$. Some consequences of this result are as follows: Every function in PFH is polynomial-time 1-Turing reducible to some function in # P. If $\mathrm{PF}^{\#\mathrm{P}[1]} \subseteq \#\mathrm{PH}$, then PH collapses to a finite level; furthermore, if either $\#\mathrm{P} \subseteq \mathrm{PFH}$ or $\mathrm{PFH} \subseteq \#\mathrm{P}$, then PH collapses to a finite level. We also give an affirmative answer to an open question posed by Valiant (1979), and we show a generalized result about p-rankability by Hemachandra (1987).

1. Introduction

Recently, researchers have been studying the computational power of *counting* in several contexts. In particular, # P [26], the class of functions that count the number of accepting paths of polynomial time-bounded nondeterministic Turing machines, has been studied, since it contains many interesting counting functions [26, 27]. The task of the present investigation is to compare # P with other classes of functions and to simplify relationships among them.

Our particular interest in this paper is to compare # P with # PH and with PFH by using some notions of polynomial-time reducibilities among functions. PFH is a function-analogue of the polynomial-time hierarchy [20, 31] (PH for short) and will later be called *the polynomial-time function hierarchy*. In many recent studies in computational complexity theory, PH has been viewed as a measure for estimating the computational power of complexity classes of sets below PSPACE. By a natural correspondence between PH and PFH, we consider PFH as a measure for estimating computational power of complexity classes of *functions* that are at least polynomial-space computable.

In addition, # PH is a generalization of # P, which is the class of functions that count the number of accepting paths of polynomial time-bounded nondeterministic oracle Turing machines with oracle sets from PH. There are many natural functions in # PH that might be outside of # P. A typical one is the function that counts different Hamiltonian subgraphs of a given graph, which was considered in [26, 10] but was not known to be complete for # NP [26] or for span-P [10] even under polynomial-time Turing reducibility. Speaking more generally, any function that, given an object such as a graph, counts different subobjects with a required property is in # PH if the property is decidable within PH; in addition, if the property does not appear to be decidable in polynomial time (for example, if it is NP-hard property), then the function appears to be out of # P. The reader may refer to the paper [10] for some concrete examples.

There have been several earlier works concerning the above classes of functions. Valiant [26, 27] showed that many natural counting functions are complete for # P under suitable polynomial-time reducibility notions. Wagner [29, 30] studied a hierarchy of classes of counting problems and Krentel [12] characterized PF^{NP} in terms of optimization functions. However, their results are characterizations of those classes but are not direct comparisons among them. (Although Krentel compared several classes below PF^{NP}, lower classes of PFH are not subject to this paper.) The only related result along this line is due to Köbler et al. [10]. They showed that # P ≠ # NP unless NP = co-NP and # P ≠ span-P unless UP = NP (for the definitions of UP and span-P, refer to the paper [25] and [10], respectively).

The present work has been motivated by the above result in [10] and a recent result in [22]. In the latter paper, it was shown that PP^{PH} is polynomial-time Turing reducible to PP; indeed, a somewhat stronger result was shown in that paper: $PP^{PH} \subseteq P^{\#P[1]}$. Currently, it is well understood that # P is polynomial-time Turing equivalent to PP and this equivalence holds between the corresponding relativized classes with respect to all oracle sets. Hence, we know that # PH is polynomial-time Turing equivalent to PP^{PH}. From these observations, we see that every function in # PH is polynomial-time Turing reducible to a function in # P; we also note that a stronger result mentioned below does not appear to be immediately obtained even if we use the stronger result in [22] (i.e. $PP^{PH} \subseteq P^{\#P[1]}$). On the other hand, from the result in [10] mentioned above, we know that if NP ≠ co-NP, # PH is *not* polynomial-time reducible to any functions in # P under the most restricted reducibility

notion, which was called *parsimonious reducibility* by Valiant [26, 27]. This is because all of the known classes in $\#$ PH are closed under that reducibility. Hence, there is a gap between Turing reducibility and parsimonious reducibility. A natural question here is whether Turing reducibility is necessary for reducing $\#$ PH to $\#$ P. As a more restricted reducibility notion than Turing reducibility, Valiant [27] and Krentel [12] used *polynomial-time* 1*-Turing reducibility*. This notion allows us to use a given function at most once as an oracle function and to make a deterministic polynomial-time computation after getting a value from the oracle function (in contrast, Turing reducibility allows us to use oracle functions as many times as possible and parsimonious reducibility does not allow us to make any computations after querying). Using this reducibility notion, our question is precisely stated as follows.

Question. Is every function in $\#$ PH polynomial-time 1-Turing reducible to a function in $\#$ P?

We give an affirmative answer to this question. That is, our main result is stated as follows.

(1) *Every function in* $\#$ PH *is polynomial-time* 1*-Turing reducible to a function in* $\#$ P, *i.e.* $\#\,\text{PH} \subseteq \text{PF}^{\#\,\text{P}[1]}$.

Recall that if NP $\neq$ co-NP, then $\#$ PH is not polynomial-time parsimonious reducible to $\#$ P. Thus, we may observe that a major gap between Turing reducibility and parsimonious reducibility exists between 1-Turing reducibility and parsimonious reducibility. In other words, if we deal with polynomial-time reductions to $\#$ P functions, then 1-Turing reductions appear to be more powerful than parsimonious reductions.

From the main result, we have the following corollaries, which are comparisons among classes of functions.

(2) *Every function in* PFH *is polynomial-time* 1*-Turing reducible to a function in* $\#$ P. *That is,* $\text{PFH} \subseteq \text{PF}^{\#\,\text{P}[1]}$.

(3) *If* $\text{PF}^{\#\,\text{P}[1]} \subseteq \#\,\text{PH}$, *then* PH *collapses to a finite level.*

(4) *If either* $\#\,\text{P} \subseteq \text{PFH}$ *or* $\text{PFH} \subseteq \#\,\text{P}$, *then* PH *collapses to a finite level; hence,* $\#$ P *and* PFH *are incomparable unless* PH *collapses.*

The result (3) above tells us that $\#$ PH is not closed under polynomial-time 1-Turing reductions unless PH collapses; on the other hand, $\#$ PH is closed under polynomial-time parsimonious reductions. Furthermore, noting that $\#$ P is closed under polynomial-time parsimonious reducibility, the result (4) above tells us that some function in PFH is not polynomial-time parsimonious reducible to any functions in $\#$ P unless PH collapses; however, we have (2).

In the rest of this paper, we show more consequences of the main result. The next consequence concerns an open question posed by Valiant [26, 27]. He asked whether the function that counts different Hamiltonian subgraphs of a given graph is complete

for # NP under a suitable reducibility although he observed that the function is polynomial 1-Turing hard for # P. (We also note that Köbler et al. [10] tried to find an affirmative answer to this question.) This question is solved affirmatively as (5), which is stated in more general form as (6).

(5) *Every function that is hard for* # P *under polynomial-time* 1*-Turing reducibility is hard for* # PH *under the same reducibility; in addition, if it is in* # PH, *then it is complete for* # PH *under polynomial-time* 1*-Turing reducibility.*

(6) *The function that counts different Hamiltonian subgraphs of a given graph is polynomial-time* 1*-Turing complete for* # NP.

We also generalize a result due to Hemachandra for *P-rankability* of sets in P. In [8], he showed that if every set in P is P-rankable, then $\mathrm{P}=\mathrm{P}^{\#\mathrm{P}}$. Here we consider Δ_k^{P}*-rankability* instead of P-rankability (for the definition of this notion, see Section 5).

(7) *For each* $k>0$, *if every set in* P *is* Δ_k^{P}*-rankable, then* $\mathrm{PH}=\Delta_k^{\mathrm{P}}=\mathrm{P}^{\#\mathrm{P}}$.

2. Preliminaries

We assume that the reader is familiar with the basic concepts of computational complexity theory. Let Σ be a finite alphabet. Our sets in this paper are over $\{0, 1, \#\}$ unless otherwise specified. Also, the symbol # is usually used as a delimiter among strings of $\{0, 1\}^*$. A pairing function (k-tuple function) over $\{0, 1\}^*$ is represented by separating two strings (k strings) by this symbol. For a string $w\in\Sigma^*$, $|w|$ denotes the length of w. For a set $L\subseteq\Sigma^*$, $\bar{L}$ denotes the complement of L. For a class $\boldsymbol{C}$ of sets, co-$\boldsymbol{C}$ denotes the class of sets whose complement is in $\boldsymbol{C}$. Let Σ^n, $\Sigma^{\leq n}$ and $\Sigma^{<n}$ denote the sets of strings with length n, at most n and less than n, respectively. For a finite set $X\subseteq\Sigma^*$, $\|X\|$ denotes the number of strings in X. Let $\mathbf{N}$ denote the set of natural numbers. We assume that all natural numbers are expressed in binary notation. Hence, for a natural number m, $|m|$ denotes the length of the binary string that expresses m.

In order to simplify our discussions, we use a special symbol $\perp\notin\Sigma$, so that for any function f we can write $f(x)=\perp$ if f is undefined on x. For any functions f and g, $f\circ g$ denotes the composition of f and g, i.e. $f\circ g(x)=f(g(x))$ for each x.

Our computational models are variations of standard Turing machines. A machine is either an acceptor or a transducer. An acceptor is deterministic or nondeterministic; on the other hand, a transducer is always deterministic. An acceptor is denoted by M or M_i, and a transducer is denoted by N or N_i. Let $L(M)$ denote the set of strings accepted by M. For any string x, we say that N *on* x *computes* y if N on input x enters an accepting state with y on its output tape. The symbol N is often used to denote the function computed by a transducer N. Note that $N(x)=\perp$ if N does not enter an accepting state.

We also consider oracle computation. An *oracle machine* may ask queries to an oracle set during its computations in a usual way. A set accepted by M relative to an oracle set A is denoted by $L(M^A)$, and a function computed by N relative to A is denoted by N^A. We consider another type of oracle machine, which we call a *function-oracle machine*. A function-oracle machine has a special tape, an *answer tape*, as well as a query tape. During its computation relative to an oracle function f, when a machine enters a query state with a string z on its query tape, it receives $f(z)$ (or, $\perp$ if $f(z)$ is undefined) on its answer tape. Here we assume that the machine consumes $|f(z)|$ steps for this process; in other words, it takes $|f(z)|$ steps to receive $f(z)$ from the oracle. For any function f, a set accepted by M relative to f is denoted by $L(M^f)$, and a function computed by N relative to f is denoted by N^f.

We abbreviate by a P-machine (NP-machine) a polynomial time-bounded deterministic (nondeterministic) machine. For any oracle machine, we use the same abbreviations.

For an oracle set A, P^A denotes the class of sets accepted by P^A-machines, i.e. oracle P-machines with oracle A. NP^A is defined similarly. Classes in the polynomial-time hierarchy relativized with oracle A are denoted in the usual way: $\Sigma_0^{\text{P},A}=\Pi_0^{\text{P},A}=\Delta_0^{\text{P},A}=\text{P}^A$, $\Sigma_k^{\text{P},A}=\text{NP}^{\Sigma_{k-1}^{\text{P},A}}$, $\Pi_k^{\text{P},A}=\text{co-}\Sigma_k^{\text{P},A}$, $\Delta_k^{\text{P},A}=\text{P}^{\Sigma_{k-1}^{\text{P},A}}$, and $\text{PH}^A=\bigcup_{k\geqslant 0}\Sigma_k^{\text{P},A}$.

$\oplus\text{P}(A)$ denotes the class of sets L for which there exists an NP^A-machine M such that for each x, x is in L iff the number of accepting computation paths of M^A on input x is odd. This class was introduced by Papadimitriou and Zachos [15].

For a class $\boldsymbol{C}$ of oracle sets, $\text{P}^{\boldsymbol{C}}=\bigcup\{\text{P}^A: A\in\boldsymbol{C}\}$. The other classes are defined similarly. The unrelativized classes are defined by setting the oracle set to the empty set, and the specification of oracle set is omitted in this case.

$\#\text{P}^A$ denotes the class of functions that give the number of accepting computation paths of NP^A-machines. For each $k\geqslant 0$, we define $\#\Sigma_k^{\text{P}}=\#\text{P}^{\Sigma_k^{\text{P}}}$ [26]. We also define $\#\text{PH}=\bigcup_{k\geqslant 0}\#\Sigma_k^{\text{P}}$. PF^A denotes the class of functions that are computable in polynomial time with oracle A. *The polynomial-time function hierarchy* are defined as follows: $\text{PFH}=\bigcup_{k\geqslant 0}\text{PF}^{\Sigma_k^{\text{P}}}$.

Let A and B be any sets. We say that A is $\leqslant_{\text{maj}}^{\text{P}}$*-reducible* to B ($A\leqslant_{\text{maj}}^{\text{P}} B$) if there exists a function f computable in polynomial time such that for each x, $f(x)=y_1\,\#\,y_2\,\#\cdots\#\,y_m$ ($m\geqslant 1$) and $x\in A$ iff the majority of y_i's are in B. We say that A is $\leqslant_{\text{T}}^{\text{P}}$*-reducible* to B ($A\leqslant_{\text{T}}^{\text{P}} B$) iff there exists a P^B-machine that accepts A. Let $\boldsymbol{C}$ be any class of sets and let $\leqslant_r^{\text{P}}$ denote any reducibility. Then we say that $\boldsymbol{C}$ *is closed under* $\leqslant_r^{\text{P}}$ if for any sets A and B, $A\leqslant_r^{\text{P}} B$ and $B\in\boldsymbol{C}$ imply $A\in\boldsymbol{C}$. It is obvious that if A is $\leqslant_{\text{maj}}^{\text{P}}$-reducible to B, then A is $\leqslant_{\text{T}}^{\text{P}}$-reducible to B. Hence, if a class $\boldsymbol{C}$ is closed under $\leqslant_{\text{T}}^{\text{P}}$, then it is closed under $\leqslant_{\text{maj}}^{\text{P}}$.

Let X be a finite set of strings and let R be a predicate over strings. In this paper, we denote by $Pr\{w\in X: R(w)\}$ the probability that $R(w)$ is true for randomly chosen w from X under uniform distribution. We define some operators that produce new classes of sets and functions from a given class of sets. These notions will be used for proving the main result.

Definition 2.1 (Schöning [17, 16]). (1) For any class $\boldsymbol{C}$ of sets, $\oplus \cdot \boldsymbol{C}$ is the class of sets L such that for some set $C \in \boldsymbol{C}$, some polynomial p, and all $x \in \Sigma^*$,

$$x \in C \leftrightarrow \| \{w \in \{0,1\}^{p(|x|)} : x \# w \in C\} \| \text{ is odd.}$$

(2) For any class $\boldsymbol{C}$ of sets, $\mathsf{BP} \cdot \boldsymbol{C}$ is the class of sets L such that for some set $C \in \boldsymbol{C}$, some polynomial p, some $\varepsilon > 0$, and all $x \in \Sigma^*$,

$$x \in L \rightarrow Pr\{w \in \{0,1\}^{p(|x|)} : x \# w \in C\} \geqslant 1/2 + \varepsilon,$$

$$x \notin L \rightarrow Pr\{w \in \{0,1\}^{p(|x|)} : x \# w \in C\} \leqslant 1/2 - \varepsilon.$$

It is obvious that $\oplus \mathrm{P} = \oplus \cdot \mathrm{P}$. So, for the sake of simplifying arguments, we will use both definitions of $\oplus \mathrm{P}$ according to context.

Definition 2.2. For any class $\boldsymbol{C}$ of sets, $\mathsf{NUM} \cdot \boldsymbol{C}$ is the class of functions f such that for some $C \in \boldsymbol{C}$, some polynomial p, and all $x \in \Sigma^*$, $f(x) = \| \{w \in \Sigma^{p(|x|)} : x \# w \in C\} \|$.

Simon [18] and Valiant [26, 27] defined the notion of parsimonious reducibility in a restricted sense that the reduction preserve the number of solutions. In this paper, we define the notion in a more general sense. In the present paper, we consider the following types of reducibilities among functions.

Definition 2.3. Let g and f be any functions from Σ^* to Σ^*.

(1) f is *polynomial-time parsimonious reducible* to g ($f \leqslant^{\mathrm{PF}}_{\mathrm{par}} g$) if there exists a total function $h \in \mathrm{PF}$ such that $f = g \circ h$. We call h a $\leqslant^{\mathrm{PF}}_{\mathrm{m}}$*-reduction* from f to g.

(2) f is *polynomial-time* 1*-Turing reducible* to g ($f \leqslant^{\mathrm{PF}}_{1\text{-}\mathrm{T}} g$) if there exists a polynomial time-bounded function-oracle transducer N that computes f with asking g at most once.

(3) f is *polynomial-time Turing reducible* to g ($f \leqslant^{\mathrm{PF}}_{\mathrm{T}} g$) if there is a polynomial time-bounded function-oracle transducer N such that $f = N^g$.

For any class $\boldsymbol{F}$ of functions, we denote by $\mathrm{PF}^{\boldsymbol{F}}$ a class of functions that are polynomial-time Turing reducible to functions in $\boldsymbol{F}$. We denote by $\mathrm{PF}^{\boldsymbol{F}[1]}$ a class of functions that are polynomial-time 1-Turing reducible to functions in $\boldsymbol{F}$.

Let $\boldsymbol{F}$ be a class of functions and $\leqslant_r$ be any reducibility defined above. We say that a function f is $\leqslant_r$*-hard* for $\boldsymbol{F}$ if every function in $\boldsymbol{F}$ is $\leqslant_r$-reducible to f. Furthermore, we say that f is $\leqslant_r$*-complete* for $\boldsymbol{F}$ if it is $\leqslant_r$-hard for $\boldsymbol{F}$ and is in $\boldsymbol{F}$. We say that $\boldsymbol{F}$ *is closed under* $\leqslant_r$*-reducibility* if for every functions f and g, $f \leqslant_r g$ and $g \in \boldsymbol{F}$ imply $f \in \boldsymbol{F}$.

3. Basic lemmas for proving main theorem

The proof of the main theorem in the next section is quite involved. Indeed, we need several lemmas for proving it. The purpose of this section is to prepare lemmas needed only for proving the main result. The reader may first skip this section, may read the proof of the main theorem, and may refer to these lemmas at the time that each of them is required.

Proposition 3.1. $\#\,\mathrm{PH}=\mathsf{NUM}\cdot\mathrm{PH}$.

Proof (*outline*). Using a standard technique, we can easily see that for each class $\boldsymbol{C}$, $\#\,\mathrm{P}^{\boldsymbol{C}}=\mathsf{NUM}\cdot\mathrm{P}^{\boldsymbol{C}}$. Hence, we have that for each $k\geqslant 1$, $\#\,\Sigma^{\mathrm{P}}_k=\#\,\mathrm{P}^{\Sigma^{\mathrm{P}}_k}=\mathsf{NUM}\cdot\Delta^{\mathrm{P}}_{k+1}$. This implies that $\#\,\mathrm{PH}=\bigcup_{k\geqslant 1}\mathsf{NUM}\cdot\Delta^{\mathrm{P}}_k=\mathsf{NUM}\cdot\mathrm{PH}$. □

Proposition 3.2 (Toda [22]). $\mathrm{PH}\subseteq\mathsf{BP}\cdot\oplus\,\mathrm{P}$.

Proposition 3.3 (Papadimitriou and Zachos [15]). $\oplus\,\mathrm{P}^{\oplus\,\mathrm{P}}=\oplus\,\mathrm{P}$. *Hence* $\oplus\,\mathrm{P}$ *is closed under* $\leqslant^{\mathrm{P}}_{\mathrm{T}}$*-reducibility.*

Proposition 3.4 (Schöning [17]). *Let* $\boldsymbol{C}$ *be a class of sets being closed under* $\leqslant^{\mathrm{P}}_{\mathrm{maj}}$. *Then for any set* $A\in\mathsf{BP}\cdot\boldsymbol{C}$ *and any polynomial* q, *there exist a set* $B\in\boldsymbol{C}$ *and a polynomial* p *such that for every* x, $Pr\{w\in\{0,1\}^{p(|x|)}: x\,\#\,w\in B \text{ iff } x\in A\}\geqslant 1-2^{-q(|x|)}$.

The following lemma is a modification of a result in [22].

Lemma 3.5. *Let* L *be a set in* $\oplus\,\mathrm{P}$ *and* q *any polynomial. Then there exists a function* $h\in\#\,\mathrm{P}$ *such that for every* x,

(1) *if* $x\in L$, *then* $h(x)\equiv 1 \pmod{2^{q(|x|)}}$, *and*

(2) *if* $x\notin L$, *then* $h(x)\equiv 0 \pmod{2^{q(|x|)}}$.

Proof. Let M be a polynomial-time bounded NTM that witnesses $L\in\oplus\,\mathrm{P}$. That is, for every x,

(i) if $x\in L$, then $\#\,\mathrm{acc}_M(x)\equiv 1 \pmod 2$, and

(ii) if $x\notin L$, then $\#\,\mathrm{acc}_M(x)\equiv 0 \pmod 2$.

To construct a function h satisfying the conditions (1) and (2) above, we first define two functions f and g as follows.

(a) $f(x)=\#\,\mathrm{acc}_M(x)^{q(|x|)}$.

(b) $g(x,0)=f(x)$, and $g(x,i)=g(x,i-1)^2+2\cdot g(x,i-1)$ for each $i>0$.

Then the function h is defined by

$$h(x)=g(x,\lceil\log_2 q(|x|)\rceil)^2.$$

First, we show that h satisfies the required conditions. After that, we will show that h is in $\#\,\mathrm{P}$. Let x be any input for M.

Claim 1. *If* $x \in L$, *then* $h(x) \equiv 1 \pmod{2^{q(|x|)}}$.

Proof of Claim 1. If $x \in L$, then $\# \operatorname{acc}_M(x) \equiv f(x) \equiv 1 \pmod 2$. We show by induction on i that

$$g(x, i) \equiv -1 \pmod{2^{2^i}} \quad \text{for each } i \geqslant 0. \tag{*}$$

The case of $i=0$ is obvious. Assume that (*) holds for every $k<i$. Hence, we have $g(x, k)=2^{2^k} \cdot m_{x,k}-1$ for all $k<i$ and some positive integer $m_{x,k}$. Then,

$$g(x, i)=(2^{2^{i-1}} \cdot m_{x,i-1}-1)^2+2(2^{2^{i-1}} \cdot m_{x,i-1}-1)=2^{2^i} \cdot m_{x,i-1}^2-1.$$

Thus, we have (*). From (*) and the definition of h, for every x, there exist positive integers m_1, m_2, and l such that

$$h(x)=(2^{2^{\lceil \log_2 q(|x|) \rceil}} \cdot m_1-1)^2=(2^{q(|x|)+l} \cdot m_1-1)^2=2^{q(|x|)} \cdot m_2+1.$$

Hence, $h(x) \equiv 1 \pmod{2^{q(|x|)}}$. □

Claim 2. *If* $x \notin L$, *then* $h(x) \equiv 0 \pmod{2^{q(|x|)}}$.

Proof of Claim 2. If $x \notin L$, then $f(x) \equiv 0 \pmod{2^{q(|x|)}}$ since $\# \operatorname{acc}_M(x)$ is even. From the definition of g, it is easy to see that for every x, every $i \geqslant 0$, and some positive integer $k_{x,i}$, $g(x, i)=f(x) \cdot k_{x,i}$. Thus, we have $g(x, i) \equiv 0 \pmod{2^{q(|x|)}}$ for every $i \geqslant 0$. □

It remains to show $h \in \# \mathrm{P}$. For the sake of simplicity, let us define a function $h_1(x)=g(x, \lceil \log_2 q(|x|) \rceil)$. To prove $h \in \# \mathrm{P}$, it suffices to show $h_1 \in \# \mathrm{P}$.

Claim 3. $h_1 \in \# \mathrm{P}$.

Proof. We first define a sequence of sets $H_i, i \geqslant 0$ over $\Gamma=\{a, b, (\ , \ )\}$ and a set G as follows.

(i) $H_0=\{a\}$,

(ii) $H_i=H_{1,i} \cup H_{2,i}$, where
$H_{1,i}=\{(w_1)(w_2): w_j \in H_{i-1}\}$ and
$H_{2,i}=\{b(w): w \in H_{i-1}\}$, and

(iii) $G=\{x \# w: x \in \Sigma^* \text{ and } w \in H_{\lceil \log_2 q(|x|) \rceil}\}$.

For each $w \in \bigcup_{k \geqslant 0} H$, we denote by $\#_a(w)$ ($\#_b(w)$) the number of a (b) occurring in w. Then we define a function $h_2: \Sigma^* \# \Gamma^* \to \boldsymbol{N}$ by

$$h_2(x \# w)=2^{\#_b(w)} \cdot f(x)^{\#_a(w)}.$$

It is easy to see that $h_2 \in \# \mathrm{P}$. Below, we show by induction on i that for every x and every $i \geqslant 0$,

$$g(x, i)=\sum_{w \in H_i} h_2(x \# w). \tag{*}$$

The case of $i=0$ is obvious. Assume that for every $k<i$, $(*)$ holds. Then,

$$\begin{aligned} g(x,i) &= g(x,i-1)^2 + 2\cdot g(x,i-1) \\ &= \left(\sum_{w\in H_{i-1}} h_2(x\#w)\right)^2 + 2\cdot \sum_{w\in H_{i-1}} h_2(x\#w) \\ &= \sum_{\langle w_1,w_2\rangle\in H_{i-1}\times H_{i-1}} h_2(x\#w_1)\cdot h_2(x\#w_2) + \sum_{w\in H_{i-1}} (2\cdot h_2(x\#w)) \\ &= \sum_{w\in H_{1,i}} h_2(x\#w) + \sum_{w\in H_{2,i}} h_2(x\#w) \\ &= \sum_{w\in H_i} h_2(x\#w). \end{aligned}$$

Hence, we have $(*)$ for all x and $i\geqslant 0$. From $(*)$ and the definitions of h_1 and G, we have that for each $x\in\Sigma^*$,

$$h_1(x) = \sum_{x\#w\in G} h_2(x\#w).$$

We note that for each $x\in\Sigma^*$ and each $w\in\Gamma^*$, if $x\#w\in G$, then $|w|$ is bounded above by a polynomial in $|x|$. Hence, if G is in P, then we can conclude $h_1\in\#\,\mathrm{P}$. However, it is not hard to see that $G\in\mathrm{P}$. The simplest way to show this is to describe a machine accepting G by either a log space-bounded alternating Turing machine [5] or a log space-bounded auxiliary pushdown automaton [6]. The detailed proof of this is left to the reader. □

The following is an immediate consequence of the above lemma.

Corollary 3.6. *Let L be a set in $\oplus\,\mathrm{P}$ and q any polynomial. Then there exists a set $A\in\mathrm{P}$ and a polynomial p such that for every x,*

(1) *if $x\in L$, then $\|\{w\in\{0,1\}^{p(|x|)}: x\#w\in A\}\|\equiv 1 \pmod{2^{q(|x|)}}$, and*

(2) *if $x\notin L$, then $\|\{w\in\{0,1\}^{p(|x|)}: x\#w\in A\}\|\equiv 0 \pmod{2^{q(|x|)}}$.*

4. The main result

In this section, we prove the main result.

Theorem 4.1. *Every function in $\#\,\mathrm{PH}$ is polynomial-time 1-Turing reducible to a function in $\#\,\mathrm{P}$. That is, $\#\,\mathrm{PH}\subseteq\mathrm{P}\Gamma^{\#\,\mathrm{P}[1]}$.*

Proof. Since $\#\,\mathrm{PH}\subseteq\mathsf{NUM}\cdot\mathrm{PH}\subseteq\mathsf{NUM}\cdot\mathsf{BP}\cdot\oplus\,\mathrm{P}$ from Propositions 3.1 and 3.2, it suffices to show that every function in $\mathsf{NUM}\cdot\mathsf{BP}\cdot\oplus\,\mathrm{P}$ is $\leqslant^{\mathrm{PF}}_{1\text{-T}}$-reducible to a function in $\#\,\mathrm{P}$. Let f be a function in $\mathsf{NUM}\cdot\mathsf{BP}\cdot\oplus\,\mathrm{P}$. Let L_1 and p_1 be a set in $\mathsf{BP}\cdot\oplus\,\mathrm{P}$ and a polynomial, respectively, that witness $f\in\mathsf{NUM}\cdot\mathsf{BP}\cdot\oplus\,\mathrm{P}$. Let L_2 and p_2 be a set in $\oplus\,\mathrm{P}$ and a polynomial, respectively, that witness $L_1\in\mathsf{BP}\cdot\oplus\,\mathrm{P}$. Furthermore, let L_3 and p_3 be a set in P and a polynomial, respectively, that witness $L_2\in\oplus\,\mathrm{P}$ (for this setting, recall that $\oplus\,\mathrm{P}=\oplus\cdot\mathrm{P}$). Summarizing the current situation above, we have

the following.

(1) For every x, $f(x)=\|\{y\in\{0,1\}^{p_1(|x|)}: x\#y\in L_1\}\|$.

(2) For some $\varepsilon>0$ and every $x\#y$ such that $|y|=p_1(|x|)$,

(a) $x\#y\in L_1\to Pr\{z\in\{0,1\}^{p_2(|x\#y|)}: x\#y\#z\in L_2\}\geqslant 1/2+\varepsilon$, and

(b) $x\#y\notin L_1\to Pr\{z\in\{0,1\}^{p_2(|x\#y|)}: x\#y\#z\in L_2\}\leqslant 1/2-\varepsilon$.

(3) For every $x\#y\#z$ such that $|y|=p_1(|x|)$ and $|z|=p_2(|x\#y|)$, $x\#y\#z\in L_2$ if and only if for an odd number of strings $w\in\{0,1\}^{p_3(|x\#y\#z|)}$, $x\#y\#z\#w\in L_3$.

Here we notice that in the above conditions, we have modified each of the original definitions so that they have become suitable for the proof of this theorem. For simplicity, we will use the following notations:

- $r_2(n)=p_2(n+1+p_1(n))$ (the length of z in (2) above).
- $l_1(n)=n+1+p_1(n)$ (the length of $x\#y$ in (1) above).
- $l_2(n)=n+1+p_1(n)+r_2(n)$ (the length of $x\#y\#z$ in (2) above).

To prove this theorem, we need several assumptions on the current situation, each of which follows from lemmas in the previous section. Since $\oplus$P is closed under $\leqslant_T^P$-reducibility from Proposition 3.3, we can apply Proposition 3.4 to the class $\mathsf{BP}\cdot\oplus\mathrm{P}$. Hence, we may assume the following.

(A1) We can take any polynomial e so that for every $x\#y$ such that $|y|=p_1(|x|)$,

(a) $x\#y\in L_1\to Pr\{z\in\{0,1\}^{p_2(|x\#y|)}: x\#y\#z\in L_2\}\geqslant 1-2^{-e(|x\#y|)}$, and

(b) $x\#y\notin L_1\to Pr\{z\in\{0,1\}^{p_2(|x\#y|)}: x\#y\#z\in L_2\}\leqslant 2^{-e(|x\#y|)}$.

We notice that the conditions in (A1) above are equivalent to a single condition that for every $x\#y$ such that $|y|=p_1(|x|)$,

(A1′) $Pr\{z\in\{0,1\}^{p_2(x\#y)}: x\#y\in L_1 \text{ iff } x\#y\#z\notin L_2\}\leqslant 2^{-e(|x\#y|)}$.

We will later use this condition instead of (A1) itself. We further notice that we can take the polynomial e independently of the polynomial p_1 (this is obvious because we may only increase the length of strings z if we need a large polynomial as e). Thus, we may make the following assumption.

(A2) The above polynomial e satisfies $2^{p_1(n)}<2^{e(l_1(n))}-1$ for every $n\geqslant 0$.
(Recall that $l_1(n)$ stands for the length of strings $x\#y$ in (A1).)

In addition, from Corollary 3.6, we may make the following assumption.

(A3) We can choose any fixed polynomial q so that for every $x\#y\#z$ such that $|y|=p_1(|x|)$ and $|z|=p_2(|x\#y|)$,

(a) $x\#y\#z\in L_2\to \|\{w\in\{0,1\}^{p_3(|x\#y\#z|)}: x\#y\#z\#w\in L_3\}\|\equiv 1 \pmod{2^{q(|x\#y\#z|)}}$

(b) $x\#y\#z\notin L_2\to \|\{w\in\{0,1\}^{p_3(|x\#y\#z|)}: x\#y\#z\#w\in L_3\}\|\equiv 0 \pmod{2^{q(|x\#y\#z|)}}$

As in the previous assumption (A2), we can take the polynomial q independently of the polynomials p_1, p_2, and e (this is also obvious because we may only increase the length of strings w in (A3) if we need a large polynomial as q). Thus, we may make the following assumption.

(A4) The above polynomial q satisfies that $p_1(n)+r_2(n)+1<q(l_2(n))$ for all $n\geqslant 0$.
(Recall that $l_2(n)$ stands for the length of strings $x\#y\#z$ in (A3) above.)

Under this setting, we define an NTM M working on an input x as follows:

Step 1: M guesses $y \in \{0,1\}^{p_1(|x|)}$, $z \in \{0,1\}^{p_2(x\#y)}$, and $w \in \{0,1\}^{p_3(|x\#y\#z|)}$.

Step 2: If $x\#y\#z\#w$ is in L_3, then M accepts the input; otherwise, M rejects the input.

Obviously, M is polynomial time-bounded. Let $\#\,\mathrm{acc}_{\mathrm{M}}(x)$ denote the number of accepting computation paths of M on input x. The purpose below is to show that we can compute $f(x)$ from $\#\,\mathrm{acc}_{\mathrm{M}}(x)$ within polynomial time in $|x|$ by using the above assumptions. More precisely, we will prove that for every x of length n,

$$f(x)=\lfloor (\#\,\mathrm{acc}_{\mathrm{M}}(x) \bmod 2^{q(l_2(n))})/(2^{r_2(n)}-2^{r_2(n)-e(l_1(n))}) \rfloor .$$

Obviously, this provides us with a polynomial-time 1-Turing reduction from f to $\#\,\mathrm{acc}_{\mathrm{M}}$, a function in $\#\,\mathrm{P}$.

The proof is done by consecutive modifications of the descriptions of $\#\,\mathrm{acc}_{\mathrm{M}}(x)$. Let x be any string of length n. First of all, by using the assumption (A3), we can express $\#\,\mathrm{acc}_{\mathrm{M}}(x)$ as follows:

$$\#\,\mathrm{acc}_{\mathrm{M}}(x)=\sum_{x\#y\in L_1}\left\{\sum_{x\#y\#z\in L_2}(2^{q(l_2(n))}\cdot k_{x\#y\#z}+1)+\sum_{x\#y\#z\notin L_2}2^{q(l_2(n))}\cdot k_{x\#y\#z}\right\}$$

$$+\sum_{x\#y\notin L_1}\left\{\sum_{x\#y\#z\in L_2}(2^{q(l_2(n))}\cdot k_{x\#y\#z}+1)+\sum_{x\#y\#z\notin L_2}2^{q(l_2(n))}\cdot k_{x\#y\#z}\right\},$$

where y and z range over $\{0,1\}^{p_1(n)}$ and $\{0,1\}^{r_2(n)}$, respectively, and each $k_{x\#y\#z}$ is a positive integer depending only on $x\#y\#z$, which comes from (A3).

Next, we define $\varepsilon_{x\#y}$ (the error probability of the string $x\#y$ in L_1) by

$$\varepsilon_{x\#y}=Pr\{z\in\{0,1\}^{p_2(|x\#y|)}\colon x\#y\in L_1 \text{ iff } x\#y\#z\notin L_2\}.$$

Notice that $\varepsilon_{x\#y}\leqslant 2^{-e(l_1(|x|))}$ for every $x\#y$ such that $|y|=p_1(|x|)$. Using this notation, we can further transform the above expression as follows:

$$\#\,\mathrm{acc}_{\mathrm{M}}(x)=\sum_{x\#y\in L_1}\left\{2^{q(l_2(n))}\cdot\left(\sum_{x\#y\#z\in L_2}k_{x\#y\#z}+\sum_{x\#y\#z\notin L_2}k_{x\#y\#z}\right)+\sum_{x\#y\#z\in L_2}1\right\}$$

$$+\sum_{x\#y\notin L_1}\left\{2^{q(l_2(n))}\cdot\left(\sum_{x\#y\#z\in L_2}k_{x\#y\#z}+\sum_{x\#y\#z\notin L_2}k_{x\#y\#z}\right)+\sum_{x\#y\#z\in L_2}1\right\}$$

$$=\sum_{x\#y\in L_1}\{2^{q(l_2(n))}\cdot k_{x\#y}+(1-\varepsilon_{x\#y})\cdot 2^{r_2(n)}\}$$

$$+\sum_{x\#y\notin L_1}\{2^{q(l_2(n))}\cdot k_{x\#y}+\varepsilon_{x\#y}\cdot 2^{r_2(n)}\}$$

$$=2^{q(l_2(n))}\cdot k_x+\sum_{x\#y\in L_1}(1-\varepsilon_{x\#y})\cdot 2^{r_2(n)}+\sum_{x\#y\notin L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)}$$

$$=2^{q(l_2(n))}\cdot k_x+f(x)\cdot 2^{r_2(n)}-\sum_{x\#y\in L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)}+\sum_{x\#y\notin L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)},$$

where

$$k_{x\#y}=\sum_{x\#y\#z\in L_2} k_{x\#y\#z}+\sum_{x\#y\#z\notin L_2} k_{x\#y\#z},$$

$$k_x=\sum_{x\#y\in L_1} k_{x\#y}+\sum_{x\#y\notin L_1} k_{x\#y}, \text{ and}$$

y and z range over $\{0,1\}^{p_1(n)}$ and $\{0,1\}^{r_2(n)}$, respectively. (Recall that $r_2(n)=p_2(|x\#y|)$, the length of the strings z above.)

From the assumption (A4), we have the following inequality:

$$\begin{aligned}
&f(x)\cdot 2^{r_2(n)}-\sum_{x\#y\in L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)}+\sum_{x\#y\notin L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)}\\
&\quad\leqslant 2^{p_1(n)}\cdot 2^{r_2(n)}+2^{p_1(n)}\cdot 2^{r_2(n)-e(l_1(n))} && \text{(since } f(x)\leqslant 2^{p_1(n)} \text{ and } \varepsilon_{x\#y}\leqslant 2^{-e(l_1(n))}\text{)}\\
&\quad\leqslant 2\cdot 2^{p_1(n)}\cdot 2^{r_2(n)} && \text{(since } 2^{p_1(n)}\cdot 2^{r_2(n)-e(l_1(n))}\leqslant 2^{p_1(n)}\cdot 2^{r_2(n)}\text{)}\\
&\quad< 2^{q(l_2(n))} && \text{(from (A4)).}
\end{aligned}$$

Thus, we know that

$$\#\,\mathrm{acc}_M(x)\equiv f(x)\cdot 2^{r_2(n)}-\sum_{x\#y\in L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)}+\sum_{x\#y\notin L_1}\varepsilon_{x\#y}\cdot 2^{r_2(n)} \pmod{2^{q(l_2(n))}}.$$

From the assumption (A1), $\varepsilon_{x\#y}\leqslant 2^{-e(|x\#y|)}$ for every $x\#y$ such that $y\in\{0,1\}^{p_1(|x|)}$. Then, we define $\varepsilon'_{x\#y}=2^{-e(|x\#y|)}-\varepsilon_{x\#y}$. Using this notation, the last congruence expression can be transformed as follows:

$$\begin{aligned}
\#\,\mathrm{acc}_M(x)&\equiv f(x)\cdot 2^{r_2(n)}\cdot(1-2^{-e(l_1(n))})\\
&\quad+2^{r_2(n)}\cdot\left(\sum_{x\#y\in L_1}\varepsilon'_{x\#y}+\sum_{x\#y\notin L_1}\varepsilon_{x\#y}\right)\pmod{2^{q(l_2(n))}}.
\end{aligned}$$

Furthermore, by using the assumption (A2),

$$\begin{aligned}
&2^{r_2(n)}\cdot\left(\sum_{x\#y\in L_1}\varepsilon'_{x\#y}+\sum_{x\#y\notin L_1}\varepsilon_{x\#y}\right)\\
&\quad\leqslant 2^{r_2(n)}\cdot\left(\sum_{x\#y\in L_1}2^{-e(l_1(n))}+\sum_{x\#y\notin L_1}2^{-e(l_1(n))}\right) && \text{(since } \varepsilon'_{x\#y},\ \varepsilon_{x\#y}\leqslant 2^{-e(l_1(n))}\text{)}\\
&\quad=2^{r_2(n)}\cdot 2^{p_1(n)}\cdot 2^{-e(l_1(n))}\\
&\quad<2^{r_2(n)}\cdot(2^{e(l_1(n))}-1)\cdot 2^{-e(l_1(n))} && \text{(from (A2))}\\
&\quad=2^{r_2(n)}\cdot(1-2^{-e(l_1(n))}).
\end{aligned}$$

From the last congruence expression and the last inequality, we have

$$f(x)=\lfloor(\#\,\mathrm{acc}_M(x) \bmod 2^{q(l_2(n))})/(2^{r_2(n)}-2^{r_2(n)-e(l_1(n))})\rfloor,$$

which is our final goal in this proof. □

5. Some consequences of the main result

In this section, we show some consequences from our main result in the previous section. We first show that every function in PFH is polynomial-time 1-Turing reducible to a function in $\#\mathrm{P}$.

Theorem 5.1. *Every function in* PFH *is polynomial-time* 1*-Turing reducible to a function in* $\#\mathrm{P}$. *That is,* $\mathrm{PFH}\subseteq\mathrm{PF}^{\#\mathrm{P}[1]}$.

Proof. From Theorem 4.1, it suffices to show that every function in PFH is polynomial-time 1-Turing reducible to a function in $\#\mathrm{PH}$. Let $f:\{0,1\}^*\to\{0,1\}^*$ be a function in some $\mathrm{PF}^{\Sigma^{\mathrm{P}}_k}$. Notice that f is not integer-valued function but all of its values are regarded as a binary representation of a natural number. Then there exist a polynomial time-bounded deterministic oracle transducer N and an oracle set $A\in\Sigma^{\mathrm{P}}_k$ such that $N^A(x)=f(x)$ for every x. Let p be a polynomial bounding the run time of N. Hence, the length of all values of f is also bounded above by the polynomial p. We define a nondeterministic oracle Turing machine M with an oracle set A as follows:

Step 1: First, M computes $w=N^A$ on a given input x.
Let m denote a natural number expressed by $1w$ in binary.

Step 2: Next, M guesses a natural number k (in binary) such that $1\leqslant k\leqslant 2^{p(|x|)+1}$.
If $k\leqslant m$, then it enters an accepting state; otherwise, it enters a rejecting state.

It is obvious that M is polynomial time-bounded and $\mathrm{bin}(\#\,\mathrm{acc}_M(x))=1w$ for every x, where $\mathrm{bin}(k)$ denotes the binary representation of a natural number k and $w=N^A(x)$. From these observations, it is easy to see that the function N^A is polynomial-time 1-Turing reducible to $\#\,\mathrm{acc}_M$, a function in $\#\,\mathrm{P}(A)$. □

Next, we present evidence for the existence of a function that is $\leqslant^{PF}_{1\text{-}T}$-reducible to $\#\,\mathrm{P}$ but is not in $\#\,\mathrm{PH}$. The following theorem is based on a technique due to Köbler et al. [10] and Theorem 5.1.

Theorem 5.2. *For every* $k\geqslant 0$, *if* $\mathrm{PF}^{\#\mathrm{P}[1]}\subseteq\#\,\Sigma^{\mathrm{P}}_k$, *then* $\mathrm{PH}=\Sigma^{\mathrm{P}}_{k+1}$.

Proof. We assume that $\mathrm{PF}^{\#\mathrm{P}[1]}\subseteq\#\,\Sigma^{\mathrm{P}}_k$. Let L be any set in PH. Let χ_L denote the characteristic function of L. Here we shall view χ_L as a function from strings to integers either 0 or 1. It is obvious that χ_L is in PFH. Hence, from the assumption and

Theorem 5.1, χ_L is in $\#\Sigma_k^{\mathrm{P}}$. Let M and $A \in \Sigma_k^{\mathrm{P}}$ be a polynomial time-bounded NOTM and an oracle set for M that witness $\chi_L \in \#\Sigma_k^{\mathrm{P}}$. Then we have that for every x, $x \in L$ iff $\chi_L(x)=1$ iff $\#\mathrm{acc}_{\mathrm{M}^A}(x)=1$ iff there exists an accepting computation path of M^A on input x. Hence, $L \in \Sigma_{k+1}^{\mathrm{P}}$. This implies $\mathrm{PH}=\Sigma_{k+1}^{\mathrm{P}}$. □

Furthermore, from the following proposition, we have a general result of Theorem 5.2.

Proposition 5.3. $\mathrm{PF}^{\#\mathrm{P}[1]}$ *has a* $\leqslant_{\mathrm{par}}^{\mathrm{PF}}$*-complete function.*

Proof. For any nondeterministic Turing machine M, we define a function $\#\mathrm{acc}_{\mathrm{M}}^b : \{0,1\}^* \times \{1\}^* \to \mathbf{N}$ as follows: for every input x to M and a natural number t, $\#\mathrm{acc}_{\mathrm{M}}^b(x, 1^t)$ = the number of accepting paths of M on x whose length is at most t. Let us define $UNIV$ as follows:

$$UNIV(N, \mathrm{M}, x, 1^t) = \begin{cases} w & \text{if } N \text{ on } x \text{ outputs } w \text{ within } t \text{ steps using} \\ & \#\mathrm{acc}_{\mathrm{M}}^b(\cdot, 1^t) \text{ at most once as an oracle function,} \\ \perp & \text{otherwise,} \end{cases}$$

where N is (an encoding of) a deterministic function-oracle transducer, M is (an encoding of) a nondeterministic Turing machine, x is an input to N, and t is a natural number. By a standard technique, it is easy to show that $UNIV$ is $\leqslant_{\mathrm{par}}^{\mathrm{PF}}$-complete for $\mathrm{PF}^{\#\mathrm{P}[1]}$. □

Corollary 5.4. If $\mathrm{PF}^{\#\mathrm{P}[1]} \subseteq \#\mathrm{PH}$, then PH collapses to a finite level.

Proof. If $\mathrm{PF}^{\#\mathrm{P}[1]} \subseteq \#\mathrm{PH}$, then a $\leqslant_{\mathrm{par}}^{\mathrm{PF}}$-complete function for $\mathrm{PF}^{\#\mathrm{P}[1]}$ is in some $\#\Sigma_k^{\mathrm{P}}$. It is easy to see that each $\#\Sigma_k^{\mathrm{P}}$ is closed under $\leqslant_{\mathrm{par}}^{\mathrm{PF}}$-reducibility. Hence, we know $\mathrm{PF}^{\#\mathrm{P}[1]} \subseteq \#\Sigma_k^{\mathrm{P}}$; hence, this corollary follows from Theorem 5.2. □

The following is immediate from Theorem 4.1 and the transitivity of $\leqslant_{1\text{-T}}^{\mathrm{PF}}$-reducibility.

Corollary 5.5. *Every function that is* $\leqslant_{1\text{-T}}^{\mathrm{PF}}$*-hard for* $\#\mathrm{P}$ *is* $\leqslant_{1\text{-T}}^{\mathrm{PF}}$*-hard for* $\#\mathrm{PH}$.

From this corollary, if a function is $\leqslant_{1\text{-T}}^{\mathrm{PF}}$-hard for $\#\mathrm{P}$ and is in $\#\mathrm{PH}$, then it is $\leqslant_{1\text{-T}}^{\mathrm{PF}}$-complete for $\#\mathrm{PH}$. In particular, we can give an affirmative answer to a question that was posed by Valiant in [26]. In that paper, he asked whether the function that counts different Hamiltonian subgraphs of a given graph is complete for $\#\mathrm{NP}$ under suitable reducibility although he observed that the function is $\leqslant_{1\text{-T}}^{\mathrm{PF}}$-hard for $\#\mathrm{P}$. We notice that Köbler et al. [10] tried to find an affirmative answer to this question.

Corollary 5.6. *The function that gives the number of different Hamiltonian subgraphs of a given graph is* $\leqslant^{PF}_{1\text{-}T}$*-complete for* $\#NP$.

From now on, we state some consequences of Theorem 5.1. Let $\#SAT$ denote the function that gives the number of satisfying assignments of a given boolean formula. It was shown in [27] that $\#SAT$ is $\leqslant^{PF}_{par}$-complete for $\#P$. We use this fact in the next corollary.

Corollary 5.7. (1) *For every* $k \geqslant 0$, *if every function in* $\#P$ *is* $\leqslant^{PF}_{T}$*-reducible to a function in* $PF^{\Sigma^p_k}$, *then* $PH = \Delta^p_{k+1} = P^{\#P}$.

(2) *If every function in* $\#P$ *is* $\leqslant^{PF}_{T}$*-reducible to a function in* PFH, *then* PH *collapses to a finite level and is equal to* $P^{\#P}$.

(3) *If either* $\#P \subseteq PFH$ *or* $PFH \subseteq \#P$, *then* PH *collapses to a finite level. Hence,* $\#P$ *and* PFH *are incomparable unless* PH *collapses to a finite level.*

(4) *If every function in* PFH *is* $\leqslant^{PF}_{par}$*-reducible to a function in* $\#P$, *then* PH *collapses to a finite level.*

Proof. (1) It is obvious that each $PF^{\Sigma^p_k}$ and $PF^{\#P}$ is closed under $\leqslant^{PF}_{T}$-reducibility. Hence, if every function in $\#P$ is $\leqslant^{PF}_{T}$-reducible to a function in $PF^{\Sigma^p_k}$ then from Theorem 5.1, we have $PF^{\Sigma^p_k} \subseteq PFH \subseteq PF^{\#P} \subseteq PF^{\Sigma^p_k}$, which implies $PH = \Delta^p_{k+1} = P^{\#P}$.

(2) Assume every function in $\#P$ is $\leqslant^{PF}_{T}$-reducible to a function in PFH. Then $\#SAT$, which is $\leqslant^{PF}_{par}$-complete for $\#P$, is in some $PF^{\Sigma^p_k}$. Since $PF^{\Sigma^p_k}$ is closed under $\leqslant^{PF}_{T}$, we have that every function in $\#P$ is $\leqslant^{PF}_{T}$-reducible to a function in $PF^{\Sigma^p_k}$. Thus, this corollary follows from (1) above.

(3) Assume $\#P \subseteq PFH$. Then, $\#SAT$ is in $PF^{\Sigma^p_k}$ for some $k \geqslant 0$ and, hence, $\#P \subseteq PF^{\Sigma^p_k}$. This implies $PH = \Delta^p_{k+1}$. On the other hand, assume $PFH \subseteq \#P$. Then, as in Theorem 5.2, we can easily see that all characteristic functions of sets in PH can be realized as $\#P$ functions whose values are either 0 or 1. This implies all sets in PH are in NP. Thus, PH collapses.

(4) This is immediate from (3). We only note that $\#P$ is closed under $\leqslant^{PF}_{par}$-reducibility. □

Next we consider a generalization of results for P-rankability of sets in P. In [8], Hemachandra showed that if every set in P is P-rankable, then P = NP. We consider here Δ^p_k-rankability as a slight generalization of P-rankability. First, we define that notion.

Definition 5.8. For each set $L \subseteq \Sigma^*$, the *ranking function*, denoted $rank_L$, of L is defined by $rank_L(x) = \| \{ y \in L : y \preceq x \} \|$ for each x, where $\preceq$ denotes the lexicographical ordering on Σ^*. A set L is Δ^p_k*-rankable* if $rank_L$ is in $PF^{\Sigma^p_{k-1}}$.

By using a technique due to Hemachandra [8], we can show that for each $k \geqslant 0$, if every set in P is Δ^p_k-rankable, then $\#P$ is included in $PF^{\Sigma^p_{k-1}}$. Hence, we have the following corollary.

Corollary 5.9. *For each* $k \leqslant 0$, *if every set in* P *is* Δ_k^{P}*-rankable, then* $\mathrm{PH} = \Delta_k^{\mathrm{P}} = \mathrm{P}^{\#\mathrm{P}}$.

References

[1] D. Angluin, On counting problems and the polynomial-time hierarchy, *Theoret. Comput. Sci.* **12** (1980) 161–173.

[2] J.L. Balcázar, R.V. Book and U. Schöning, The polynomial-time hierarchy and sparse oracles, *J. Assoc. Comput. Mach.* **33** (1986) 603–617.

[3] R. Beigel, L.A. Hemachandra and G. Wechsung, On the power of probabilistic polynomial time: $\mathrm{P}^{\mathrm{NP}[\log]} \subseteq \mathrm{PP}$, in: *Proc. 4th IEEE Conf. on Structure in Complexity Theory* (1989) 225–227.

[4] Jin-yi Cai and L.A. Hemachandra, On the power of parity polynomial time, in: *Proc. Symp. on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Vol. 349 (Springer, Berlin, 1989) 229–240.

[5] A. Chandra, D. Kozen and L.J. Stockmeyer, Alternation, *J. ACM* **28** (1980) 114–133.

[6] S.A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, *J. ACM* **18** (1971) 4–18.

[7] J. Gill, Computational complexity of probabilistic Turing machines, *SIAM J. Comput.* **6** (1977) 675–695.

[8] L.A. Hemachandra, On ranking, in: *Proc. 2nd IEEE Conf. on Structure in Complexity Theory* (1987) 103–117.

[9] K. Ko, Some observations on the probabilistic algorithms and NP-hard problems, *Inform. Process. Lett.* **14** (1982) 39–43.

[10] J. Köbler, U. Schöning and J. Toran, On counting and approximation, *Acta Informatica* **26** (1989) 363–379.

[11] J. Köbler, U. Schöning, S. Toda and J. Toran, Turing machines with few accepting computations and low sets for PP, in: *Proc. 4th IEEE Conf. on Structure in Complexity Theory* (1989) 208–215.

[12] M. Krentel, The complexity of optimization problems, *J. Comput. System Sci.* **36** (1988) 490–509.

[13] C. Lauteman, BPP and the polynomial-time hierarchy, *Inform. Process. Lett.* **14** (1983) 215–217.

[14] T.J. Long and A.L. Selman, Relativizing complexity classes with sparse oracles, *J. ACM* **33** (1986) 618–627.

[15] C.H. Papadimitriou and S. Zachos, Two remarks on the power of counting, in: *Proc. 6th GI Conf. on TCS*, Lecture Notes in Computer Science, Vol. 145 (Springer, Berlin, 1983) 276–296.

[16] U. Schöning, The power of counting, in: *Proc. 3rd IEEE Conf. on Structure in Complexity Theory* (1988) 2–9.

[17] U. Schöning, Probabilistic complexity classes and lowness, in: *Proc. 2nd IEEE Conf. on Structure in Complexity Theory* (1987) 2–8; also *J. Comput. System Sci.* **39** (1989) 84–100.

[18] J. Simon, On the difference between one and many, in: *Proc. 4th Coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 52 (Springer, Berlin, 1977) 480–491.

[19] M. Sipser, A complexity theoretic approach to randomness, in: *Proc. 15th ACM Symp. on Theory of Computing* (1983) 330–335.

[20] L.J. Stockmeyer, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977) 1–22.

[21] L.J. Stockmeyer, On approximation algorithms for $\#\mathrm{P}$, *SIAM J. Comput.* **14** (1985) 849–861.

[22] S. Toda, On the computational power of PP and $\oplus\mathrm{P}$, in: *Proc. 30th IEEE Symp. on Foundations of Computer Science* (1989) 514–519.

[23] S. Toda, Restricted relativizations of probabilistic polynomial time, *Theoret. Comput. Sci.*, accepted for publication (January 1990).

[24] J. Toran, An oracle characterization of the counting hierarchy, in: *Proc. 3rd IEEE Conf. on Structure in Complexity Theory* (1988) 213–223.

[25] L.G. Valiant, Relative complexity of checking and evaluating, *Inform. Process. Lett.* **5** (1976) 20–23.

[26] L.G. Valiant, The complexity of computing the permanent, *Theoret. Comput. Sci.* **8** (1979) 189–201.

[27] L.G. Valiant, The complexity of reliability and enumeration problems, *SIAM J. Comput.* **8** (1979) 410–421.

[28] L.G. Valiant and V.V. Vazirani, NP is as easy to detecting unique solutions, *Theoret. Comput. Sci.* **47** (1986) 85–93.
[29] K. Wagner, The complexity of combinatorial problems with succinct input representation, *Acta Informatica* **23** (1986) 325–356.
[30] K. Wagner, Some observations on the connection between counting and recursion, *Theoret. Comput. Sci.* **47** (1986) 131–147.
[31] C. Wrathall, Complete sets and the polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977) 23–33.
[32] S. Zachos, Robustness of probabilistic computational complexity classes under definitional perturbations, *Inform. and Control* **54** (1982) 143–154.
[33] S. Zachos and H. Heller, A decisive characterization of BPP, *Inform. and Control* **69** (1986) 125–135.